12-2008

Approximations to Continuous Processes in Hierarchical Models

Amanda Cangelosi
Utah State University

APPROXIMATIONS TO CONTINUOUS PROCESSES IN HIERARCHICAL MODELS

by

Amanda R. Cangelosi

A thesis submitted in partial fulfillment
of the requirements for the degree

of

MASTER OF SCIENCE

in

Statistics

Approved:

Dr. Mevin Hooten Major Professor	Dr. Richard Cutler Committee Member
Dr. Brynja Kohler Committee Member	Dr. Byron R. Burnham Dean of Graduate Studies

UTAH STATE UNIVERSITY
Logan, Utah

2008

ABSTRACT

Approximations to Continuous Processes

in Hierarchical Models

by

Amanda R. Cangelosi, Master of Science

Utah State University, 2008

Major Professor: Dr. Mevin Hooten
Department: Mathematics and Statistics

Models for natural nonlinear processes, such as population dynamics, have been given much attention in applied mathematics. For example, species competition has been extensively modeled by differential equations. Often, the scientist has preferred to model the underlying dynamical processes (*i.e.*, theoretical mechanisms) in continuous-time. It is of both scientific and mathematical interest to implement such models in a statistical framework to quantify uncertainty associated with the models in the presence of observations. That is, given discrete observations arising from the underlying continuous process, the unobserved process can be formally described while accounting for multiple sources of uncertainty (*e.g.*, measurement error, model choice, and inherent stochasticity of process parameters). In addition to continuity, natural processes are often bounded; specifically, they tend to have non-negative support. Various techniques have been implemented to accommodate non-negative processes, but such techniques are often limited or overly compromising. This study offers an alternative to common differential modeling practices by using a bias-corrected truncated normal distribution to model the observations and

latent process, both having bounded support. Parameters of an underlying continuous process are characterized in a Bayesian hierarchical context, utilizing a fourth-order Runge-Kutta approximation.

(54 pages)

ACKNOWLEDGMENTS

I would like to thank my advisor, Mevin Hooten, for his extensive mentoring and endless enthusiasm. I also thank Chris Wikle and Peter Adler for many helpful comments and suggestions, as well as Dave Brown for valuable technical assistance, and Brynja Kohler and Brian Yurk for mathematical insights and resources.

Amanda Cangelosi

CONTENTS

LIST OF TABLES

LIST OF FIGURES

INTRODUCTION

Continuous Models of Bounded Systems

The dynamics of natural systems have been extensively studied by theoretical scientists and applied mathematicians to describe the underlying processes (*i.e.*, mechanisms) assumed to drive the phenomena of interest (*e.g.*, population dynamics). For various scientific and philosophical reasons, the underlying theoretical mechanism is often assumed to be continuous in time, and thus the scientist assigns one of numerous possible continuous process models, usually involving differential equations, to describe the dynamical system of interest (*e.g.*, Turchin, 2003, pp. 93-108). This study addresses the need to quantify uncertainty associated with such mathematical models, highlighting the advantage of a statistical approach. Statistical implementations of mathematical models for describing dynamical ecological systems are appearing more frequently in the literature (*e.g.*, Wikle, 2003; Buckland et al., 2007; Hooten and Wikle, 2007; Gianni, Pasquali, and Ruggeri, 2008; Johnson et al., 2008), though they are often coarsely discretized. While there are situations that naturally warrant the use of discrete models (*e.g.*, matrix models, Caswell, 2001), this study focuses on the implementation of continuous models to describe underlying processes, even when the corresponding observations may be discrete.

In addition to continuity, natural processes are often bounded, meaning that their quantification is restricted to some subset of the real number line. Almost exclusively, natural processes are measured in quantities of mass, thus restricting the continuous process model (as well as the observations) to non-negative realizations. Numerous techniques have been utilized to accommodate issues of bounded support (*e.g.*, log-transformations), but often these techniques misrepresent system dynam-

ics or fail to accommodate complicated non-linear models. When data contain zeros as realizations (*e.g.*, species competition; rainfall data; rare species counts), further complications arise. A common but *ad hoc* approach is to add a small number so that a log-transformation can be taken. While these practices are widespread, they often do not represent the dynamics well. To accommodate the occurrence of zeros in data, discrete probability models (*e.g.*, Poisson) and zero-inflated models are commonplace. However, discrete probability models are reasonable only when the support of the data is discrete; this is not often the case for many measurements of mass (*e.g.*, plant basal area quadrat cover). Zero-inflated models are reasonable only when a natural model exists for the zero process; however, in many situations, zero data clearly belong to the same dynamical process as the data with positive support. This study offers an alternative method of accommodating bounded data containing zeros through the incorporation of a bias-corrected truncated normal model.

Objective

We focus on a general methodology that one may use to quantify the uncertainty associated with a given continuous model, as well as introduce a bias-corrected truncated normal distribution as a general and robust model for the data and underlying dynamical process. The truncated normal distribution is particularly desirable over the gamma or lognormal distributions, as it allows truncation boundaries to occur anywhere along the real line; specifically, the truncated normal distribution allows zero-valued realizations to have non-zero density. We show that within a hierarchical framework (Berliner, 1996), a bias-corrected truncated normal data model can be utilized to accommodate observations having bounded support. In addition, a similar bias-corrected truncated normal physical process model is introduced to describe both single-species population growth (logistic growth) and interspecies competition

(Lotka-Volterra equations, *e.g.*, Edelstein-Keshet, 1988, pp. 224-231). Although the example presented here contains non-negative integer observations, we implement the truncated normal data model (which has continuous support) to demonstrate its general utility.

We also utilize a fourth-order Runge-Kutta (RK4) approximation to the system of differential equations in conjunction with Markov chain Monte Carlo (MCMC) methods to implement the formal statistical model. While a high-order approximation, such as RK4, is not new to statistics (*e.g.*, Rumelin, 1982; Shoji and Ozaki, 1998; Scipione and Berliner, 1993), its implementation within a hierarchical framework is underutilized. RK4 is a useful approximation, well-known for its accuracy, efficiency, robustness, and applicability to dynamical processes of any finite dimension.

To illustrate our methods, we use a historical data set (Gause, 1934) containing population measurements of *Paramecium aurelia* and *Paramecium caudatum* grown in nutrient medium both separately and together. We focus on a subset of these data; specifically, we use data from days 2 through 19 provided in Table 3 of Appendix I of Gause's *The Struggle for Existence* (pp. 144-145). While this particular data set has been used in numerous studies (Leslie, 1957; Edelstein-Keshet, 1988; Pascual and Kareiva, 1996; Lele, Dennis, and Lutscher, 2007), often serving as a textbook example for biological logistic growth models, many previous studies utilize low-order discretizations, wherein population size is exclusively modeled at times coinciding with those of the observed data. In contrast, we augment the process so that it can be modeled at arbitrarily small time increments. It should be noted that the focus here is concerned mainly with the methods, while the data are used primarily for illustration.

In the following section, the truncated normal distribution is briefly described and the necessity of a bias-correction within the hierarchical framework is explained.

The process model, wherein RK4 approximation is utilized, is outlined and followed by a discussion of the entire hierarchical model. The Results section includes details about model fits pertaining to both simulated data and the Gause data. The Discussion section provides brief interpretations of results from the classical Gause data, followed by a detailed assessment of the methodology in general, a comparison to other work, and possible extensions.

METHODS

The Truncated Normal Distribution

If $X \sim N(\mu, \sigma^2)$ and $I \subset R$ is an interval, then $P(X \leq x | X \in I)$, defined for any real x, is the truncated normal distribution of X (adapted from Rohatgi, 1976, p. 116). If a normal distribution is truncated, the probability mass from the tail(s) is dispersed throughout the remaining portion. Thus, if a normal density is left-truncated at zero, then the new mode resulting from truncation is necessarily located in one of two places: If the original (untruncated) mode was positive, then the mode of the left-truncated density is unchanged (though the expectation is not); if the original mode was negative, then the mode of the left-truncated density is zero. This characteristic becomes important in justifying our choice of parameter estimation.

The truncated normal distribution can accommodate random variables with non-negative support, but a more appropriate model can be specified by implementing a bias correction in the truncated normal with a target expectation. To illustrate the suitability of such a bias correction, first suppose $X \sim \text{T.N.}(\mu, \sigma^2)$, where we desire $E(X) = \mu > 0$. The actual expectation is not μ however, but rather

$$E(X) = \mu + \sigma \left(\frac{\phi(\frac{b_1 - \mu}{\sigma}) - \phi(\frac{b_2 - \mu}{\sigma})}{\Phi(\frac{b_2 - \mu}{\sigma}) - \Phi(\frac{b_1 - \mu}{\sigma})} \right), \tag{1}$$

where b_1 and b_2 are the lower and upper bounds of truncation, and ϕ and Φ are the standard normal pdf and cdf, respectively (Johnson, Kotz, and Balakrishnan, 1994; Horrace, 2005). That is, the expectation is not equivalent to the parameter μ, as

desired. In particular, when $b_1 = 0$ and $b_2 = \infty$, we have

$$E(X) = \mu + \sigma \left(\frac{\phi\left(\frac{\mu}{\sigma}\right)}{\Phi\left(\frac{\mu}{\sigma}\right)} \right). \tag{2}$$

Though simpler than that in (1), the expectation in (2) is a complicated nested integral equation that is analytically intractable; however, it can be approximated numerically. If we specify $X \sim \text{T.N.}(g, \sigma^2)$, where g is a function of μ and σ^2 such that: $E(X) = \mu$, then we need only find the bias-correcting function g. That is, for $b_1 = 0$ and $b_2 = \infty$, one can approximate a truncated normal distribution with a target expectation μ by solving for the function g such that

$$E(X) = g + \sigma \left(\frac{\phi\left(\frac{g}{\sigma}\right)}{\Phi\left(\frac{g}{\sigma}\right)} \right) = \mu. \tag{3}$$

In addition to obtaining an appropriate distribution within the hierarchical model (*i.e.*, one that obtains the target expectation), the implementation of this bias correction also decreases process uncertainty (*i.e.*, standard deviation of the distribution), particularly near truncation bounds, as often appears in data sets and is depicted in Figure 1.

Constructing the Process Model

Let $\frac{d\mathbf{x}}{dt} = h(t, \mathbf{x})$ be a differential equation (or system of differential equations) that models a continuous dynamical process $\mathbf{x}$ in time t. Such a process can be well-approximated by the classical fourth-order Runge-Kutta method (Burden and Faires,

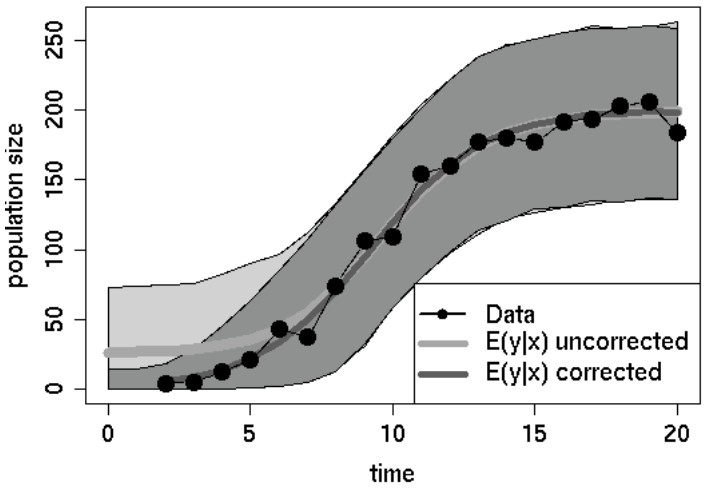


Fig. 1: *Effect of bias correction on uncertainty.* Assume a truncated normal data model, left-truncated at zero. The dark area depicts a probability envelope for a data model containing a bias correction; the light area is that for a model without a bias correction. Note the shift in the mean and reduction of uncertainty in the dark envelope near the truncation boundary.

2001):

$$\mathbf{x}_{t+\Delta t} = \mathbf{x}_t + \frac{\Delta t}{6}(a_1 + 2a_2 + 2a_3 + a_4), \tag{4}$$

$$\equiv f(\mathbf{x}_t, \boldsymbol{\theta}) \tag{5}$$

where $a_1 = h(t, \mathbf{x}_t)$, $a_2 = h(t + \frac{\Delta t}{2}, \mathbf{x}_t + \frac{\Delta t}{2}a_1)$, $a_3 = h(t + \frac{\Delta t}{2}, \mathbf{x}_t + \frac{\Delta t}{2}a_2)$, $a_4 = h(t + \Delta t, \mathbf{x}_t + a_3\Delta t)$, and $\boldsymbol{\theta}$ is a vector of parameters. Thus, the process at time $t + \Delta t$ is determined by the process at time t plus the product of the size of the time interval (Δt) and an estimated weighted average of slopes ($\frac{a_1+2a_2+2a_3+a_4}{6}$), where a_i is an estimated slope at one of four points within the time interval Δt. Note that the approximation for $\mathbf{x}_{t+\Delta t}$ is computed using four points within the time interval; this accounts for the accuracy of RK4, as the error at each of the four steps is on the order of $(\Delta t)^5$ and has an overall error on the order of $(\Delta t)^4$.

The RK4 approximation, which we have labeled $f(\mathbf{x}_t, \boldsymbol{\theta})$, can then be embedded within a hierarchical probability model at the process level. The process is "augmented" in the following way: If the data consist of an $l \times m \times n$ array (*e.g.*, m variables observed at n points in time, replicated l times), then the process will have dimenson $l \times m \times (\frac{n}{\Delta t} + 1)$, where $0 < \Delta t < 1$. Note that, depending on the value of Δt, the process can be approximated to a desired level of precision. If the process is allowed to be stochastic at each discretization time t, while Δt is specified to be small, then a state-space model with Markov dependence and an augmented process component results; *i.e.*, $\mathbf{x}_t|\mathbf{x}_{t-\Delta t}, \boldsymbol{\theta} \sim [\mathbf{x}_t|f(\mathbf{x}_{t-\Delta t}, \boldsymbol{\theta})]$, where temporal resolution of $\mathbf{x_t}$ is finer than that of the data.

In our case, $f(\mathbf{x}_{t-\Delta t}, \boldsymbol{\theta})$ is the RK4 approximation to the Lotka-Volterra model for two-species competition:

$$\frac{dx_1}{dt} = r_1 x_1 \frac{k_1 - x_1 - \beta_1 x_2}{k_1}, \tag{6}$$

$$\frac{dx_2}{dt} = r_2 x_2 \frac{k_2 - x_2 - \beta_2 x_1}{k_2}, \tag{7}$$

where the parameter vector $\boldsymbol{\theta}$ contains r_1, k_1, β_1, r_2, k_2, and β_2, the parameters controlling the dynamics of the system, each of which is assumed to exist within the interval $(0, \infty)$. Specifically, for the i^{th} species, x_i is the underlying continuous process of population growth, r_i is the growth rate, k_i is the carrying capacity, and β_i is the competition parameter. Regarding competition parameters, β_1 denotes the deleterious effect of species 2 on species 1, and vice versa. Note that in the absence of one species, this particular process model defaults to logistic growth of the other species, and the species competition model can be generalized to any number of species (Edelstein-Keshet, 1988, pp. 231-236). Further note that there is a wide variety of systems of differential equations in the literature with which one can model a given

continuous population process, each having strengths and weaknesses (Turchin, 2003).

The Hierarchical Representation

To introduce the hierarchical model, we begin with the model for observations (*i.e.*, likelihood). Let observations for the process under study at time $t = 1, \ldots, n$ be denoted by $\mathbf{y}_t$, an $l \times m \times n$ data array, where l is the number of replicates, m is the number of species observed, and n is the number of times each species is observed. Then a probability model can be written for these measurements:

$$\mathbf{y}_t|\mathbf{x}_t, \sigma^2_{\mathbf{y}} \sim \text{T.N.}(\mathbf{g}_{\mathbf{y}_t}, \mathbf{\Sigma}_{\mathbf{y}})^{b_2}_{b_1}, \ \mathbf{\Sigma}_{\mathbf{y}} = \sigma^2_{\mathbf{y}}\mathbf{I}, \tag{8}$$

where $\mathbf{g}_{\mathbf{y}_t}$ is a bias correction such that $E(\mathbf{y}_t) = \mathbf{x}_t > \mathbf{0}$, and b_1, b_2 are the lower and upper truncation boundaries, respectively. We focus on the expectation (rather than the median or mode) for two reasons: First, the mean is a natural parameter for unconstrained models and it is convenient for reparameterization; second, if the uncorrected mode is negative, then the corrected mode is necessarily zero. Note that we assigned the same variance ($\sigma^2_{\mathbf{y}}$) to each species at the data level because we expect consistent measurement error between species; we could have easily allowed the variance of each species at the data level to be different.

The data model in (8) is conditional on the second level in the hierarchy, the process $\mathbf{x}_t$. Let us adopt the formulation described in Section 2.2 as a model for the process:

$$\mathbf{x}_t|\mathbf{x}_{t-\Delta t} \sim \text{T.N.}(\mathbf{g}_{\mathbf{x}_t}, \mathbf{\Sigma}_{\mathbf{x}})^{b_2}_{b_1}, \tag{9}$$

$$\boldsymbol{\Sigma}_{\mathbf{x}} = \begin{pmatrix} \sigma^2_{x_1} & 0 \\ 0 & \sigma^2_{x_2} \end{pmatrix},$$

where $\boldsymbol{\Sigma}_{\mathbf{x}}$ is a diagonal covariance matrix, and $\mathbf{g}_{\mathbf{x}_t}$ is a bias-correcting function (dependent upon $f(\mathbf{x}_{t-\Delta t}, \boldsymbol{\theta}), \sigma^2_{\mathbf{x}}, b_1,$ and b_2) such that $E(\mathbf{x}_t) = f(\mathbf{x}_{t-\Delta t}, \boldsymbol{\theta}) > \mathbf{0}$, the RK4 approximation to the Lotka-Volterra species competition model at time t.

The latent variables $\mathbf{x}_t$ can be thought of as unobserved state vectors and may occur at a finer temporal resolution than the data, $\mathbf{y}_t$. In other words, if $\boldsymbol{\tau}_{\mathbf{x}}$, $\boldsymbol{\tau}_{\mathbf{y}}$ are finite sets of times, then $|\{\mathbf{x}_t, \forall t \in \boldsymbol{\tau}_{\mathbf{x}}\}| > |\{\mathbf{y}_t, \forall t \in \boldsymbol{\tau}_{\mathbf{y}}\}|$, where "$|\cdot|$" represents the size of the set. This point is critical for obtaining a precise approximation to the motivating system of differential equations, because in practice we could choose the set $\boldsymbol{\tau}_{\mathbf{x}}$ to be large enough to attain any desired precision in the approximation.

The parameter model makes up the third and final level of the hierarchy, controlling the dynamics of the system (whence, $\boldsymbol{\theta}$) as well as additional stochasticity (whence, $\sigma^2_{\mathbf{x}}$, $\sigma^2_{\mathbf{y}}$, the uncertainty associated with the process and data models, respectively).

$$\boldsymbol{\theta} \sim \text{T.N.}(\boldsymbol{\mu}_{\boldsymbol{\theta}}, \boldsymbol{\Sigma}_{\boldsymbol{\theta}})_{\mathbf{d_1}}^{\mathbf{d_2}}$$

$$\mathbf{x}_0 \sim \text{T.N.}(\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0)_{b_1}^{b_2}$$

$$\sigma^2_{\mathbf{y}} \sim \text{Inverse Gamma}(r_{\mathbf{y}}, q_{\mathbf{y}})$$

$$\sigma^2_{\mathbf{x}} \sim \text{Inverse Gamma}(r_{\mathbf{x}}, q_{\mathbf{x}})$$

The truncated normal distribution for $\boldsymbol{\theta}$ and $\mathbf{x}_0$ are desirable because the distribution is both flexible and proper, guaranteeing a proper posterior. The initial state vector $\mathbf{x}_0$ accommodates uncertainty in the process at the time period before data are available. In the case where Δt is small, this initial random state vector resembles the

process at the time when data are first available; thus we specify its prior distribution to have a location near that of the data where $t = 1$ and a diagonal covariance matrix with variance components larger than the variance in the data at any time point. Locational hyperparameters pertaining to the dynamic nature of the process (i.e., $\boldsymbol{\mu}_{\boldsymbol{\theta}}$) can be specified to provide reasonable dynamics while still allowing for a wide range of behavior. Specifically, it is possible to fully specify the covariance matrices $\boldsymbol{\Sigma}_{\boldsymbol{\theta}}$ and $\boldsymbol{\Sigma}_0$ to allow for prior dependence between parameters, but in the absence of specific knowledge about the form of dependence, we allowed them to be diagonal with standard deviations proportional to the magnitudes of the location parameters (*e.g.*, larger prior variability for larger parameters such as k_1 and k_2).

The inverse-gamma priors were specified to be conservative in location and wide in variability. Variance components can be modeled in different ways (Gelman, 2006), though the specification used here yielded a stable algorithm and performed well. In hierarchical models without significant structure at the process level, identifiability issues may arise with the variance components (Dennis et al., 2006). In this specification, substantial structure is provided by the latent dynamical system, thus helping partition the sources of uncertainty.

Model Implementation and Inference

The hierarchical model described above can be implemented in an MCMC setting in the usual manner (see Appendix C) by analytically identifying proportional full-conditional distributions (listed below) for parameters and latent state vectors, and then sampling from them sequentially.

$$[\mathbf{x}_0|\cdot] \propto [\mathbf{x}_1|\mathbf{x}_0, \boldsymbol{\theta}, \sigma^2_{\mathbf{x}}][\mathbf{x}_0]$$

$$[\mathbf{x}_t|\cdot] \propto \prod_{i=1}^{l} [\mathbf{y}_{i,t}|\mathbf{x}_t, \sigma^2_{\mathbf{y}}]^{\mathrm{I}_{\{t\in\tau_{\mathbf{y}}\}}} [\mathbf{x}_{t+\Delta t}|\mathbf{x}_t, \boldsymbol{\theta}, \sigma^2_{\mathbf{x}}][\mathbf{x}_t|\mathbf{x}_{t-\Delta t}, \boldsymbol{\theta}, \sigma^2_{\mathbf{x}}]$$

$$[\mathbf{x}_T|\cdot] \propto \prod_{i=1}^{l} [\mathbf{y}_{i,T}|\mathbf{x}_T, \sigma^2_{\mathbf{y}}][\mathbf{x}_{T-\Delta t}, \boldsymbol{\theta}, \sigma^2_{\mathbf{x}}]$$

$$[\boldsymbol{\theta}|\cdot] \propto \prod_{t\in\tau_{\mathbf{x}}} [\mathbf{x}_t|\mathbf{x}_{t-\Delta t}, \boldsymbol{\theta}, \sigma^2_{\mathbf{x}}][\boldsymbol{\theta}]$$

$$[\sigma^2_{\mathbf{y}}|\cdot] \propto \prod_{i=1}^{l} [\mathbf{y}_{i,t}|\mathbf{x}_t, \sigma^2_{\mathbf{y}}]^{\mathrm{I}_{\{t\in\tau_{\mathbf{y}}\}}} [\sigma^2_{\mathbf{y}}]$$

$$[\sigma^2_{\mathbf{x}}] \propto \prod_{t\in\tau_{\mathbf{x}}} [\mathbf{x}_t|\mathbf{x}_{t-\Delta t}, \boldsymbol{\theta}, \sigma^2_{\mathbf{x}}][\sigma^2_{\mathbf{x}}]$$

In this case, the truncated distributions imply nonconjugacy in all parameters and a Metropolis approach must be taken in the MCMC algorithm. Additionally, although the bias corrections are functions of $\{\mathbf{x}_t\}$, they are analytically intractable and thus must be approximated numerically, using one of several possible numerical optimization routines to find the solution.

To illustrate our numerical method of choice for the univariate case, suppose we desire a truncated normal random variable y to have expectation μ. Recall that, without a bias correction, the expectation would be as specified in (1). Thus, we need to approximate the function g such that $E(y)$ appears as in (10). Then each time that a bias-corrected truncated normal density needs to be evaluated in the MCMC algorithm, we can numerically find g conditional on μ, σ^2, b_1, b_2. This bias correction is, in effect, a reparameterization that results in a more appropriate probability model. As described below, this method is generalized for the multivariate case.

We employ a linear interpolation method (see Appendix A for details and an example) in order to exploit efficient matrix operations in the R programming environment (R Core Development Team, 2007) and find that it is precise in approximating the well-behaved function g for our purposes here. That is, in general, if we

desire a truncated normal random variable $\mathbf{y}$ to have expectation $\boldsymbol{\mu}$, then we can find a reparameterization $g(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, using the method described in Appendix A, such that $E(\mathbf{y}) = \boldsymbol{\mu}$, for $\mathbf{y}|\boldsymbol{\mu}, \boldsymbol{\Sigma} \sim$ T.N.$(g(\boldsymbol{\mu}, \boldsymbol{\Sigma}), \boldsymbol{\Sigma})$, where $\boldsymbol{\Sigma}$ is a diagonal matrix. Performance of numerical algorithms vary by situation and thus should be experimented with for best results in terms of precision and efficiency.

RESULTS

In fitting the model described in the previous section to data, we actually fit three separate models: One univariate logistic growth model to each of the single-species data, followed by a competition model using the two-species data. The singe-species models were implemented by setting competition parameters equal to zero, in which case the Lotka-Volterra equations default to logistic growth. In this case, since populations were observed both in isolation and in competition (independently), the advantage is that posterior distributions from the single-species models can be used to inform prior distributions for germane parameters (*i.e.*, r_1, r_2, k_1, k_2) in the two-species model. Thus, single-species posterior means of r_1 k_1, r_2, and k_2 were used within the competition model to then obtain estimates of the competition parameters, β_1 and β_2. This use of independent data to gain *a priori* scientific knowledge about the process is a strength of the Bayesian approach and allows us to focus all available power on the estimation of the additional competition parameters (*i.e.*, β_1, β_2). We set $\Delta t = \frac{1}{4}$, yielding an augmented process with four times the temporal resolution of the data and a very precise approximation to the differential equations. The augmentation level of $\frac{1}{4}$ was chosen because it represented the process well for this scale in the simulated data; it was large enough for computational efficiency, while providing an accurate approximation. Higher-order systems may require a smaller Δt depending on the dynamics. The MCMC algorithms were run for 20,000 iterations with a burn-in of 2,000 chosen by study of trace plots (see Appendix B).

To assess the model's capability in situations similar to the Gause data, the model was evaluated using simulated data first; the results of this are summarized in Table 1. Figures 2-5 depict results of the model, applied to Gause's *Paramecium* data. Figure

2 shows the data for single-species logistic growth, overlaid with the augmented process 95% credible intervals. With regard to the model fit using the Gause data, Figure 3 shows posterior distributions for single-species logistic growth rates and carrying capacities. Posterior mean single-species growth rates (r_1 and r_2) were 0.6762 and 1.068 with standard deviations 0.1684 and 0.5109, for *P. aurelia* and *P. caudatum*, respectively. Here, the prior distributions were $[r_1] = [r_2] = \text{TN}(0.5, 100)_{d_{\mathbf{r}_1}}^{d_{\mathbf{r}_2}}$, having truncation bounds $d_{\mathbf{r}_1} = 0$ and $d_{\mathbf{r}_2} = \infty$. Posterior mean single-species carrying capacities (k_1 and k_2) were 564.4 and 202.2, with standard deviations 26.12 and 24.15, for *P. aurelia* and *P. caudatum*, respectively. The prior distributions for the carrying capacities were $[k_1] = \text{TN}(600, 100000)_{d_{\mathbf{k}_1}}^{d_{\mathbf{k}_2}}$ and $[k_2] = \text{TN}(200, 100000)_{d_{\mathbf{k}_1}}^{d_{\mathbf{k}_2}}$, each with truncation bounds $d_{\mathbf{k}_1} = 0$, and $d_{\mathbf{k}_2} = \infty$.

Figure 4 shows the data for two-species competition, overlaid with the augmented process credible intervals, after using the single-species growth rate and carrying capacity parameters to inform the two-species priors. Figure 5 shows posterior distributions for competition parameters (β_1 and β_2). Posterior means of the competition parameters were 2.553 and 0.4742 with standard deviations 1.152 and 0.2916, for *P. aurelia* and *P. caudatum*, respectively. Here, the prior distributions were$[\beta_1] = [\beta_2] = \text{TN}(1, 10)_{d_{\boldsymbol{\beta}_1}}^{d_{\boldsymbol{\beta}_2}}$, having truncation bounds $d_{\boldsymbol{\beta}_1} = 0$ and $d_{\boldsymbol{\beta}_2} = \infty$. The hyperparameters for both single-species and competition model variance components were $r_{\mathbf{y}} = r_{\mathbf{x}} = 10^{-3}$ and $q_{\mathbf{y}} = q_{\mathbf{x}} = 2.01$, yielding $E(\sigma_{\mathbf{y}}^2) = E(\sigma_{\mathbf{x}}^2) = 1000$ and $\text{Var}(\sigma_{\mathbf{y}}^2) = \text{Var}(\sigma_{\mathbf{x}}^2) = 10^8$.

Table 1: Results from simulated data

Parameter	Truth	Posterior Mean	Posterior S.D.	Prior Mean	Prior S.D.
Single-Species 1					
r_1	0.5	0.5313	0.1473	0.5	10
k_1	600	598.8	45.45	650	316.2
Single-Species 2					
r_2	0.3	0.3547	0.1360	0.5	10
k_2	400	419.7	41.67	350	316.2
Species Competition					
r_1	0.5	0.4873	0.0821	0.5313	0.1473
k_1	600	604.6	47.59	598.8	45.45
β_1	2	1.989	0.5363	1	3.162
r_2	0.3	0.2945	0.1260	0.3547	0.1360
k_2	400	410.8	42.42	419.7	41.67
β_2	0.5	0.5870	0.3336	1	3.162

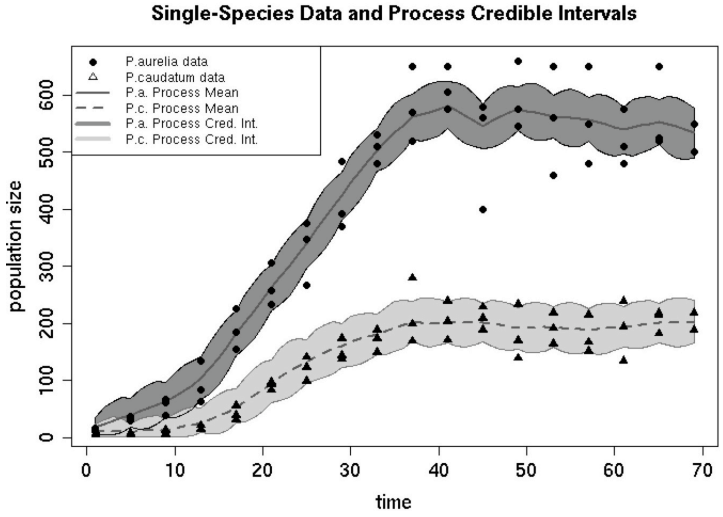
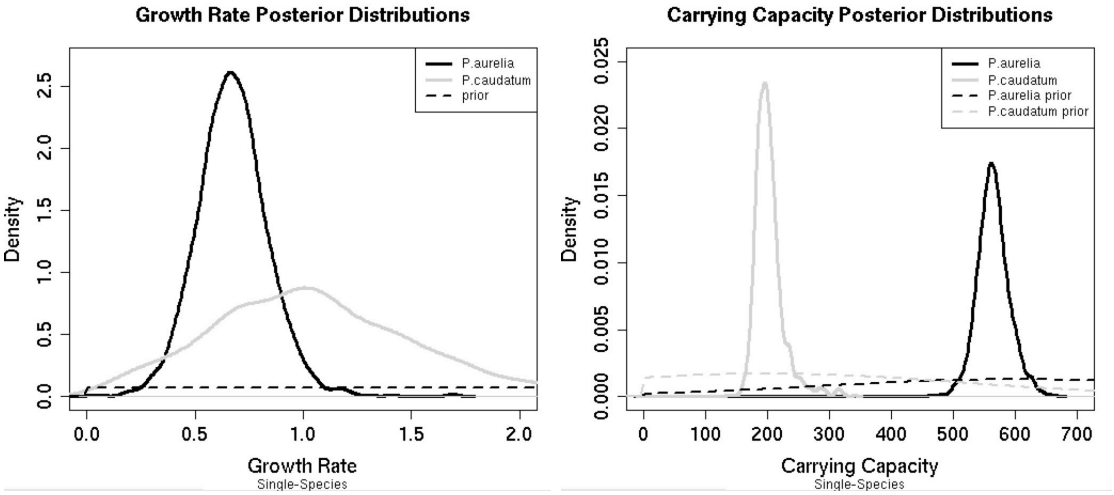


Fig. 2: *Paramecium aurelia* and *Paramecium caudatum* observations with augmented process 95% credible interval overlaid. Each species was grown in its own medium, independent of the other species (no competition).



(a) (b)

Fig. 3: Posterior distributions for single-species logistic growth rates (a) and carrying capacities (b), with prior distribution overlaid.

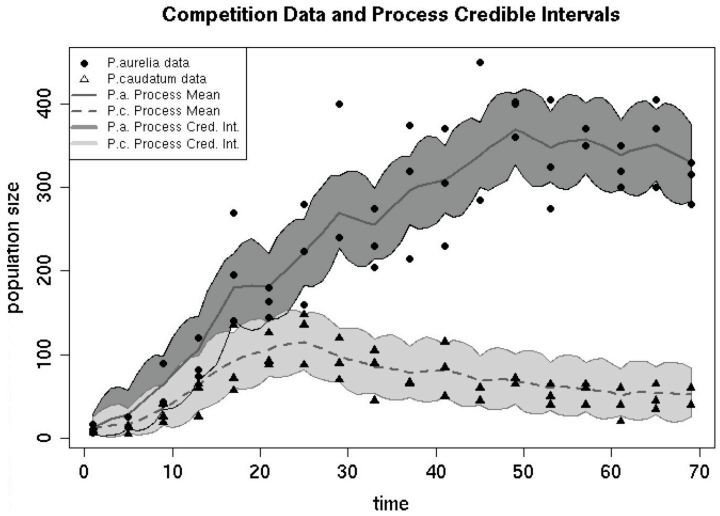


Fig. 4: Mixed *P. aurelia* and *P. caudatum* observations with augmented process 95%credible intervals overlaid. Here, each species was grown in the presence of the other species (inter-species competition).

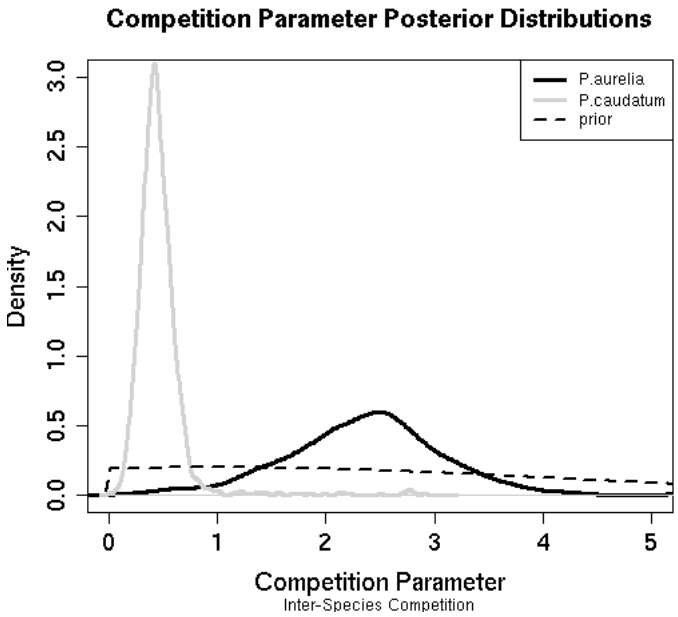


Fig. 5: Posterior distributions for competition parameters, with prior distribution overlaid.

DISCUSSION

Simulation and Gause Data

Overall, model fits suggested that the data contain significant information about the process and parameters. More specifically, regarding the simulated data summarized in Table 1, note that the posterior variance is reduced for the growth rates in the competition model, as compared to the single-species cases. This suggests that the competition data hold additional information about the single-species growth parameters. Further, note that each of the parameters used for simulation is indeed captured by the model, suggesting that the model describes the simulated process accurately. Regarding the Gause data, the overlap in posterior growth rates for single-species (Figure 3a) provides little evidence of a significant difference, although the variability differs. The fact that the posterior variability of r_2 is greater than that of r_1 may seem surprising since the *P. caudatum* observations appear to have less spread than the *P. aurelia* observations. Perhaps the difference in variability is due to the fact that *P. caudatum* initially experiences a delayed growth relative to *P. aurelia*, yet *P. caudatum* appears to meet its carrying capacity earlier than *P. aurelia*. Figure 3b suggests that *P. aurelia* is naturally capable of attaining a larger population size than *P. caudatum* in the absence of the other species. Figure 5 evinces a substantial difference among competition parameters; perhaps *P. caudatum* is more affected as a percentage of its carrying capacity. One important aspect with regard to the augmented process is that, similar to prediction uncertainty in all statistical models, the credible intervals of the augmented process indicate increased uncertainty at time points farther away from data, as seen in Figures 2 and 4.

General Methodology

Regarding the methodology of our model, the proposed bias-corrected truncated normal models alleviate the need for log-transformations and allow us to model a non-negative process. It should be noted that both the process and data models have bounded support, whereas other studies (*e.g.*, Stein, 1992) have modeled bounded data which were assumed to arise from a measurement model conditioned on a process with real support, rendering no need for a bias correction. The implementation of a bias correction allows us to specify an appropriate physical process model with positive support. Furthermore, utilizing a finer temporal discretization in the process ($\Delta t < 1$) improves stability of the dynamical system and avoids drawbacks pertaining to the representation of dynamics that accompany many analytically discretized models (*e.g.*, Ricker growth, Turchin, 2003). The process augmentation, resulting from $\Delta t < 1$, also allows our approximation to be faithful to the motivating continuous dynamical model; that is, as $\Delta t \to 0$, our model preserves the dynamical properties of the differential equations. Additionally, in the presence of stochasticity, as $\Delta t \to 0$, our model converges to an underlying stochastic differential process. The incorporation of RK4 for continuous model approximation is useful for emphasizing parameter estimation and ensuring that process uncertainty is focused on model choice rather than model approximation.

It should be noted that, as Gause's *Paramecium* data are counts, a Poisson data model would also be reasonable. We have found that the implementation of such a model (not presented here) yields similar results in terms of the posterior process and parameters. Although the Poisson specification addresses the discrete nature of the data in this case, it imposes a distinct mean-variance relationship. An overdispersed Poisson model could also be used, though we focus on the more general truncated normal model which also implies a mean-variance relationship, yet it is more flexi-

ble than the Poisson due to its two-parameter model formulation. Furthermore, the model we use here is also more robust in that it allows for linear transformations of the data (*e.g.*, scaling) as well as various types of data; for example, the proposed model is applicable to data having bounded continuous support (*e.g.*, rainfall data; percent quadrat cover; plant basal area). Application to such examples is the subject of ongoing research.

REFERENCES

Berliner, L. (1996). Hierarchical bayesian time-series models. In Hanson, K. and Silver, R., editors, *Maximum Entropy and Bayesian Methods*, pages 15–22. Kluwer Academic Publishers, Dordrecht.

Buckland, S., Newman, K., Fernandez, C., Thomas, L., and Harwood, J. (2007). Embedding population dynamics models in inference. *Statistical Science* **22,** 44–58.

Burden, R. and Faires, J. (2001). *Numerical Analysis*. Wadsworth Group, Pacific Grove, seventh edition.

Caswell, H. (2001). *Matrix Population Models: Construction, Analysis, and Interpretation*. Sinauer Associates, Sunderland.

Dennis, B., Ponciano, J., Lele, S., Taper, M., and Staples, D. (2006). Estimating density dependence, process noise, and observation error. *Ecological Monographs* **76,** 323–341.

Edelstein-Keshet, L. (1988). *Mathematical Models in Biology*. McGraw-Hill, Inc., New York.

Gause, G. (1934). *The Struggle for Existence*. The Williams and Wilkins Company, Baltimore.

Gelman, A. (2006). Prior distributions for variance parameters in hierarchical models (comment on article by browne and draper). *Bayesian Analysis* **1,** 515–534.

Gianni, G., Pasquali, S., and Ruggeri, F. (2008). Bayesian inference for functional response in a stochastic predator-prey system. *Bulletin for Mathematical Biology* **70,** 358–381.

Hooten, M. and Wikle, C. (2007). A hierarchical bayesian non-linear spatio-temporal model for the spread of invasive species with application to the eurasian collard-dove. *Environmental and Ecological Statistics* **15,** 59–70.

Horrace, W. (2005). Some results on the multivariate truncated normal distribution. Department of Economics, Syracuse University Working Paper Series.

Johnson, D., London, J., Lea, M., and Durban, J. (2008). Continuous-time correlated random walk models for animal movement data. *Ecology* **89,** 1208–1215.

Johnson, N., Kotz, S., and Balakrishnan, N. (1994). *Continuous Univariate Distributions*. John Wiley and Sons, New York.

Lele, S., Dennis, B., and Lutscher, F. (2007). Data cloning: easy maximum likelihood estimation for complex ecological models using bayesian markov chain monte carlo methods. *Ecology Letters* **10,** 551–563.

Leslie, P. (1957). An analysis of the data for some experiments carried out by gause with populations of the protozoa, paramecium aurelia and paramecim caudatum. *Biometrika* **44,** 314–327.

Pascual, M. and Kareiva, P. (1996). Predicting the outcome of competition using experimental data: Maximum likelihood and bayesian approaches. *Ecology* **77,** 337–349.

R Core Development Team (2007). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0.

Rohatgi, V. (1976). *An Introduction to Probability Theory and Mathematical Statistics*. John Wiley and Sons, New York.

Rumelin, W. (1982). Numerical treatment of stochastic differential equations. *SIAM Journal on Numerical Analysis* **19,** 604–613.

Scipione, D. and Berliner, L. (1993). Bayesian inference in nonlinear dynamical systems. In *Proceedings of the Section on Bayesian Statistical Sciences*. American Statistical Association.

Shoji, I. and Ozaki, T. (1998). A statistical method of estimation and simulation for systems of stochastic differential equations. *Biometrika* **85,** 240–243.

Stein, M. (1992). Prediction and inference for truncated spatial data. *American Statistical Association, Institute of Mathematical Statistics, and Interface Foundation of North America* **1,** 91–110.

Turchin, P. (2003). *Complex Population Dynamics: A Theoretical/Empirical Synthesis*. Princeton University Press, Princeton.

Wikle, C. (2003). Hierarchical bayesian models for predicting the spread of ecological processes. *Ecology* **84,** 1382–1394.

APPENDICES

NUMERICAL BIAS CORRECTION

This appendix describes the numerical method that we used to approximate the bias correction (*i.e.*, reparameterization) necessary for truncated normal random variables within the data and process models of our hierarchical model, both of which have non-negative support. Recall the following equation of the expectation of a random variable $X \sim \text{T.N.}(g, \sigma^2)_0^\infty$:

$$E(X) = g + \sigma \left(\frac{\phi\left(\frac{g}{\sigma}\right)}{\Phi\left(\frac{g}{\sigma}\right)} \right) = f(g, \sigma). \tag{10}$$

Let $f_0(g, \sigma)$ represent a target expectation (*e.g.*, in our hierarchical model, we desire the expectation of the data model at time t to be equivalent to the process at time t; $E(\mathbf{y}_t) = \mathbf{x}_t$). As (10) is an intractable integral equation, we cannot solve for g given f; however, we can determine f given g. Thus, we specify values of the function $f(g, \sigma)$, say $f_i(g_i, \sigma)$ for $i = \{1, \ldots, n\}$. Next, we determine which pair of $f_i(g_i, \sigma)$s flank our target value $f_0(g, \sigma)$. That is, after we identify numerous g_is, we identify g_l and g_u (respectively, values of g that are less than and greater than the target g_0 corresponding to the targeted expectation $f_0(g, \sigma)$). The points g_l and g_u form a line secant to the $f(g, \sigma)$ curve such that $f(g_l)$ and $f(g_u)$ closely approximate f_0. Via linear interpolation of the secant line, $f_0(g, \sigma)$ and hence g_0 is approximated. Finally, the approximation of g_0 is incorporated into the MCMC algorithm. See Figure 6 for an illustration of the approximation method.

The following program (*getgvec.R*) was utilized to approximate g_0 in the R programming environment.

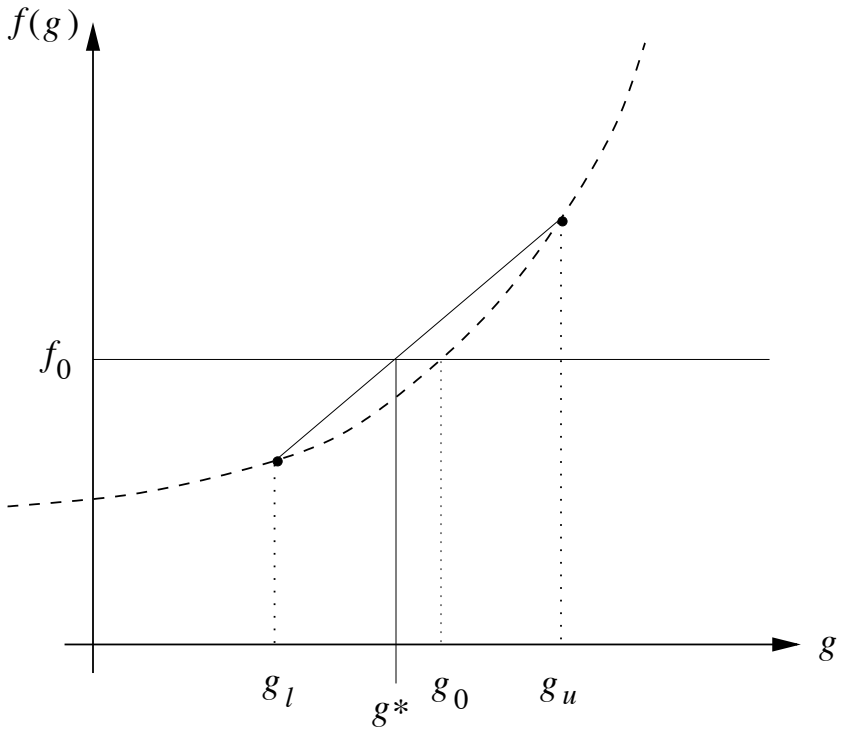


Fig. 6: Linear interpolation method used to approximate the bias correction (g_0), given target expectation f_0. Here, g^* is the approximation of g_0, and g_u and g_l define the secant line that most closely approximates the curve $f(g)$ near g_0.

```
getgvec <- function(f,sig,n){
   ### Finds reparameterized g (parameter in y~TN(g,sig) such that E(y)=f
   ###  (left-truncated at zero)
   ### n is the number of g values that are plotted, consecutive pairs of
   ### which are assessed for their proximity to the target g

  l=-100                      ### lower bound for g values to be plotted
  N=length(f)
  gtmp=seq(l,max(f),,n)       ### g values to be plotted
  G=gtmp+sig*(exp(dnorm(gtmp/sig,log=TRUE)-pnorm(gtmp/sig,log=TRUE)))
       ### f value corresponding to g value;
       ###   we seek the pair of G values that flank the target f.

  while(min(G)>=min(f)){
    l=l-100
    gtmp=c(l,gtmp)
    n=n+1
    G=gtmp+sig*(exp(dnorm(gtmp/sig,log=TRUE)-pnorm(gtmp/sig,log=TRUE)))
  } ### This loop decreases the lower bound l if corresponding G values
    ###  did not flank the target f.

  diffs=outer(as.vector(G),as.vector(f),FUN="-")
  diffs.pos=diffs
  diffs.pos[diffs < 0]=NA
  diffs.neg=diffs
  diffs.neg[diffs >= 0]=NA
    ### diff.pos and diffs.neg will be used to find G values closest to
    ###    the target f.

  idxu.mat=(diffs.pos==t(apply(diffs.pos,2,min,na.rm=TRUE)%x%matrix(1,1,n)))
  idxl.mat=(diffs.neg==t(apply(diffs.neg,2,max,na.rm=TRUE)%x%matrix(1,1,n)))
  idxu.mat[is.na(idxu.mat)]=FALSE
  idxl.mat[is.na(idxl.mat)]=FALSE

  G1=(G%x%matrix(1,1,N))[idxl.mat]      ### best G value less than target f
  G2=(G%x%matrix(1,1,N))[idxu.mat]      ### best G value greater than target f
  g1=(gtmp%x%matrix(1,1,N))[idxl.mat]   ### best g value less than target g
  g2=(gtmp%x%matrix(1,1,N))[idxu.mat]   ### best g value greater than target g

  beta1=(G2-G1)/(g2-g1)                 ### slope of secant line
  g=(f-G1)/beta1+g1                     ### linear interpolation of target g

  plot(gtmp,G,type="l",lwd=2,col=2)     ### plots G as a function of g values
  abline(h=f)                           ### labels target f
  abline(v=0)
  points(g,f,pch=20)                    ### labels coordinate point (g,f)
                                        ### (identifies g approximation)

  G=g+sig*exp(dnorm(g/sig,log=TRUE)-pnorm(g/sig,log=TRUE)))
                                        ### approximation of target f
```

```
  cbind(g,G)
}
```

The following output results from an implementation of *getgvec.R* which may be relevant to species competition such as the Gause data. Suppose one desires a target population size expectation of 1 for Species A and 2 for Species B at time t, and a standard deviation of 32 for each species. Here, there will be 100 `g` values proposed, with which to perform the interpolation:

```
> getgvec(c(1,2),32,100)
              g         G
[1,] -1022.0028 0.9999992
[2,]  -508.0508 1.9998625
```

Figure 7 displays the corresponding plot. Since the target expectations for each species are relatively close to the truncation boundary of zero, the approximations for the bias corrections **g** are negative. Note the accuracy of the interpolated mean value (**G**).

Now suppose that the mean expectations for each species were larger (*i.e.*, further from the zero-truncation boundary). For example, consider the target mean population size for Species A is 200 and that of Species B is 500, maintaining the relatively small standard deviation of 32:

```
> getgvec(c(200,500),32,100)
      g   G
[1,] 200 200
[2,] 500 500
```

Figure 8 displays the corresponding plot. Note the nature of the curve $G(g)$; since the lower and upper truncation boundaries are 0 and ∞, respectively, the curve is asymptotically linear (approaching the line $G = g$). In the specific example depicted

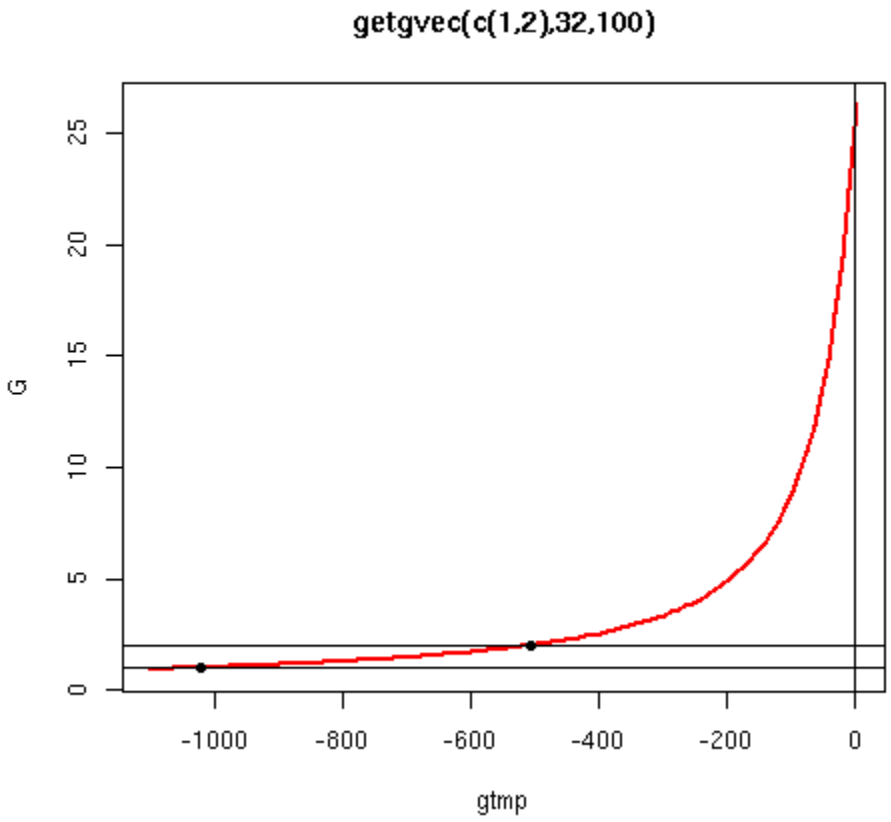


Fig. 7: Plot from getgvec(c(1,2),32,100), demonstrating the numerical approximation method of bias correction **g**. The red curve is created from Equation (1), by which 100 (g, G) points were plotted. The two horizontal lines mark the approximation of G, the corresponding approximation to the target mean **f** labeled by the points.

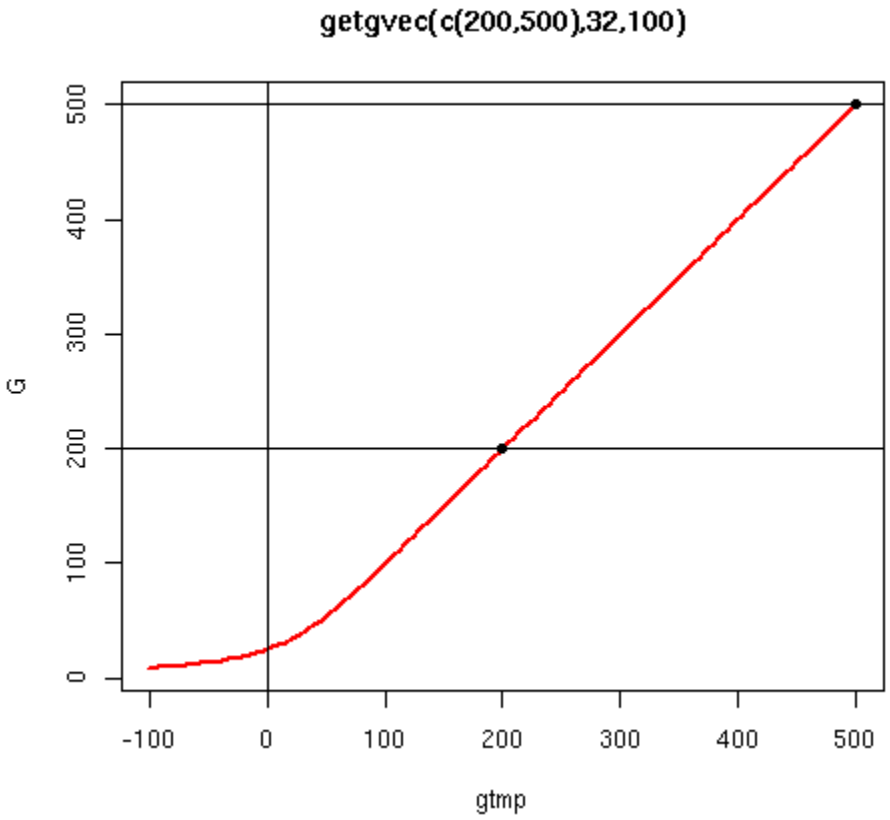


Fig. 8: Plot from getgvec(c(200,500),32,100)

in Figure 8, the approximated expectations are very accurate, due to the fact that the target expectation is relatively far from the truncation boundary of zero, coupled with a relatively small standard deviation. In the next example, we see how the approximation for the bias correction **g** and resulting approximation for the target mean **G** change when the standard deviation is increased:

```
> getgvec(c(200,500),320,100)
              g        G
[1,] -179.5233 199.9989
[2,]  447.8295 499.9949
```

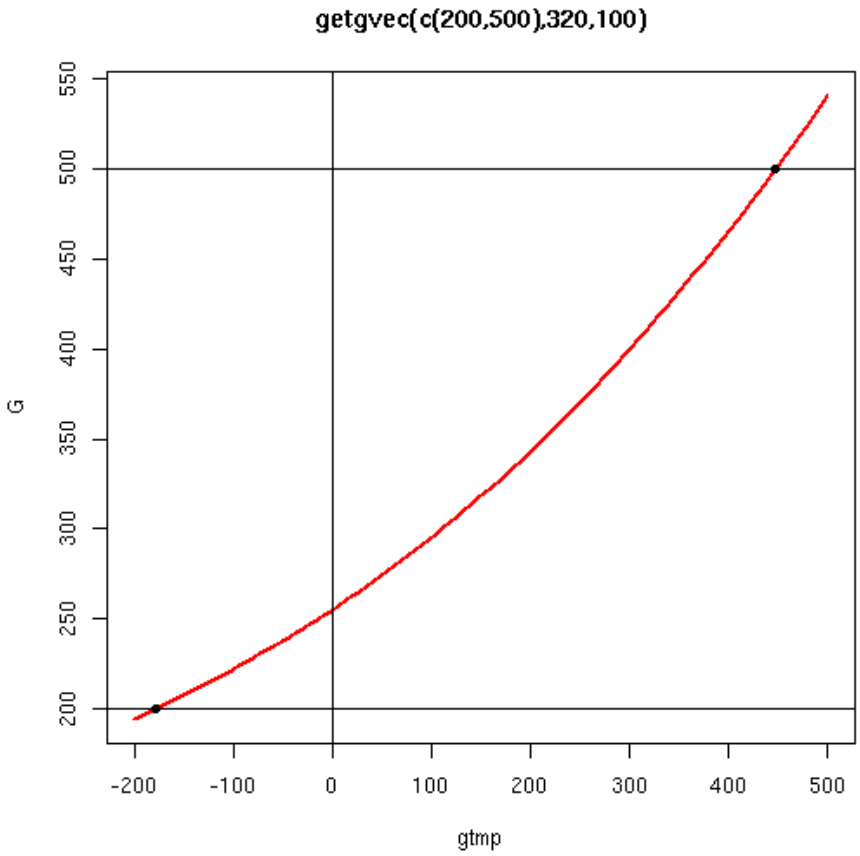


Fig. 9: Plot from getgvec(c(200,500),320,100)

Figure 9 displays the corresponding plot.

We have found this numerical approximation method to be highly efficient and accurate in matrix languages such as R and MATLAB. However, if one implements *getgvec.R* with a target expectation near zero and a large standard deviation, the method becomes inefficient. The source of such potential inefficiency is that g values become very negative very quickly, and thus it becomes difficult to identify g_l. We emphasize that, while this numerical method performs well in the settings discussed here, one may choose any preferred routine, such as a root-finding algorithm or the optimization of an objective function. For example, if $\boldsymbol{\mu}$ is a vector of target means, one may desire to minimize the objective function $Q = (E(\mathbf{X}) - \boldsymbol{\mu})^T (E(\mathbf{X}) - \boldsymbol{\mu})$ with respect to $\mathbf{g}$ (recall that $E(\mathbf{X})$ is a function of the bias-correcting function $\mathbf{g}$).

We have presented this appendix as a detailed explanation of the reparameterization of the truncated normal model in the situation where an underlying process has non-negative support. We have found the resulting bias-corrected and bounded distribution to be especially useful when modeling dynamical systems involving mass (*e.g.*, populations; hydrological processes).

TRACE PLOTS

This appendix provides trace plots of the Lotka-Volterra parameters (*i.e.*, growth rate (r_1, r_2), carrying capacity (k_1, k_2), and competition parameters (β_1, β_2)) from the Markov chain Monte Carlo algorithm. For each parameter, two trace plots are provided; one plot is obtained from all 20,000 iterations and the other is obtained from resampling every 1000 iterations.

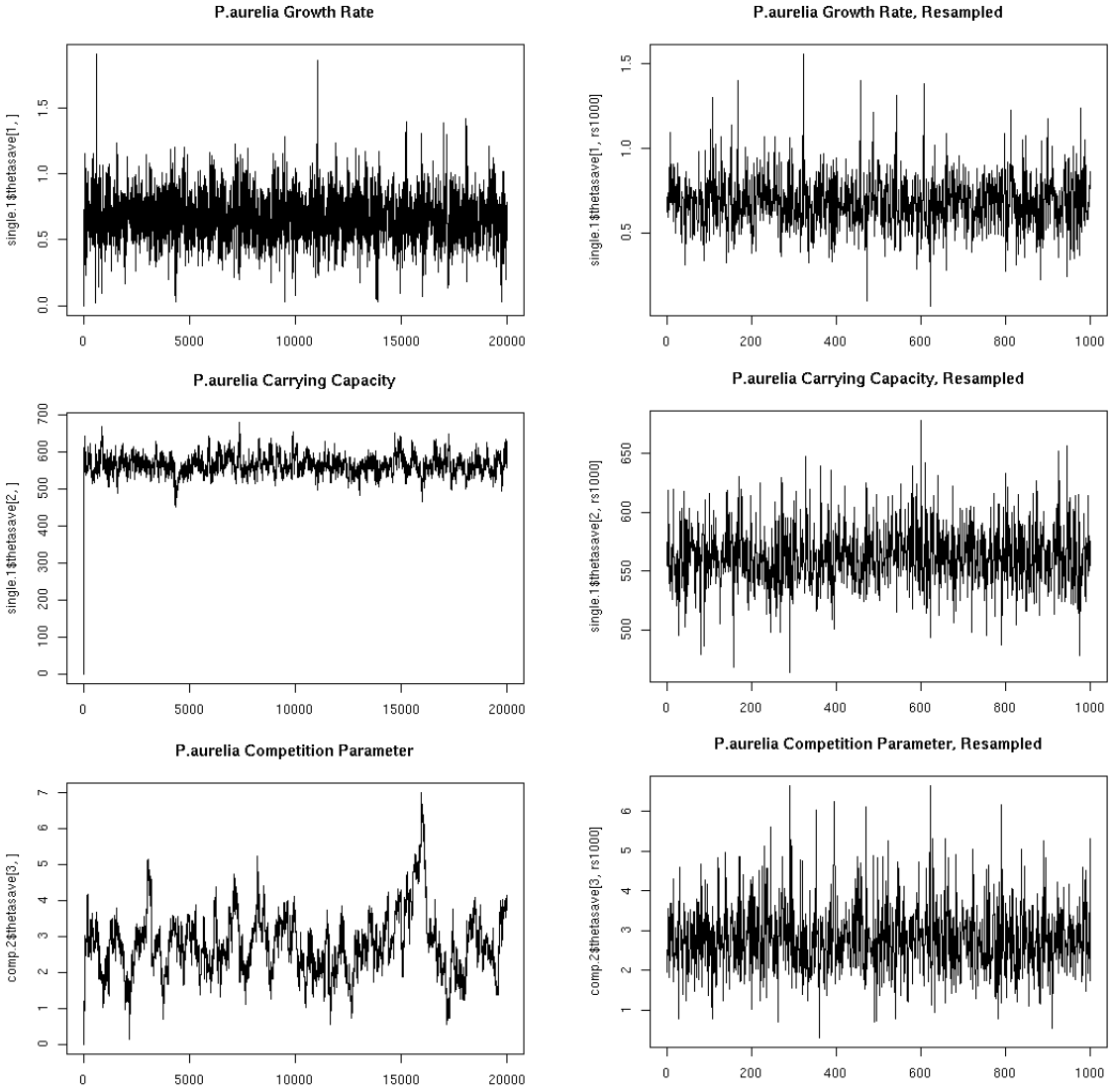


Fig. 10: Trace plots for Lotka-Volterra parameters for *P. aurelia*. The plots on the right-hand-side are resampled.

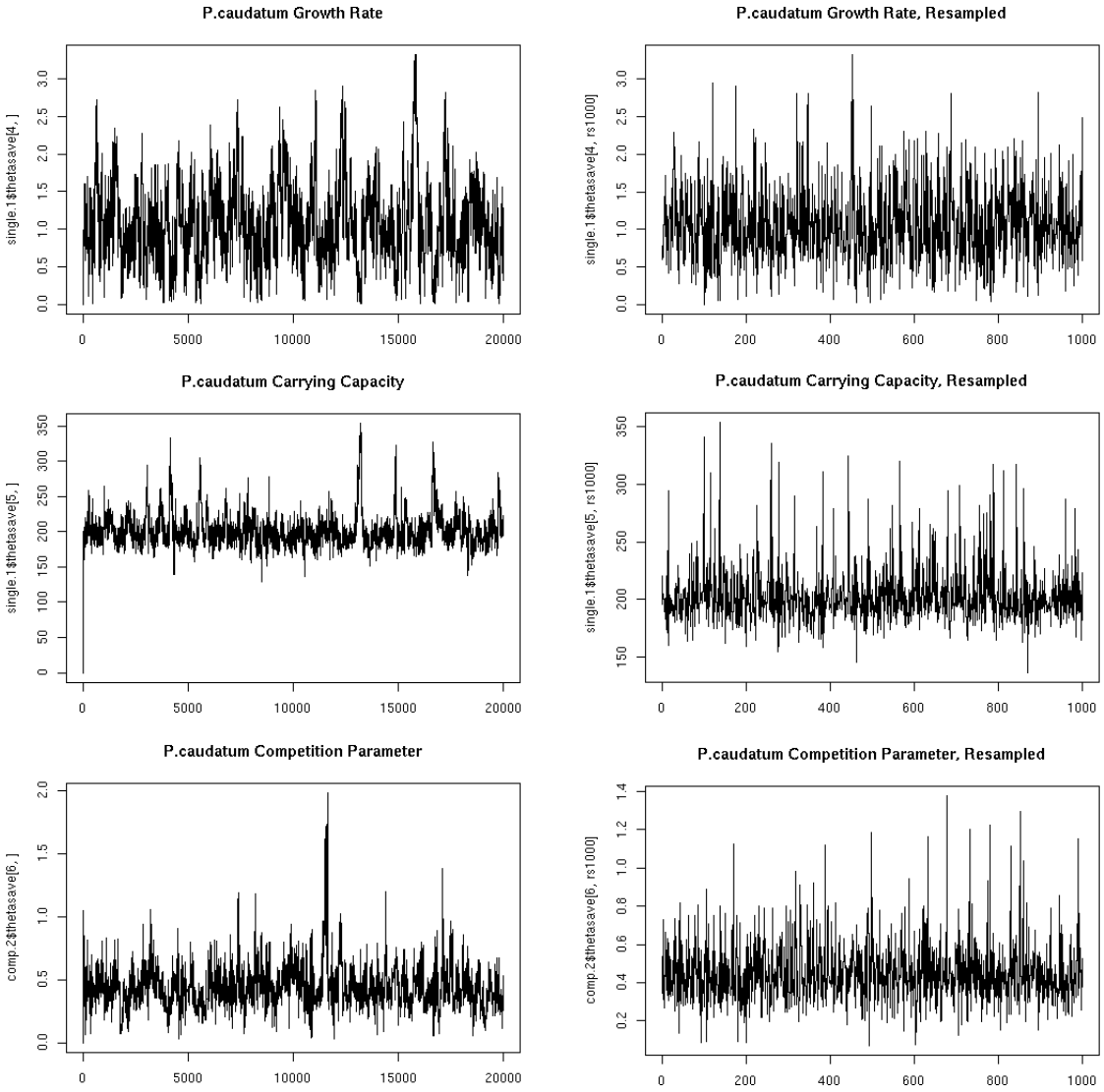


Fig. 11: Trace plots for Lotka-Volterra parameters for *P. caudatum*. The plots on the right-hand-side are resampled.

R CODE

This appendix contains the MCMC algorithm written by Mevin Hooten for an R programming environment that was used in this study. This particular example is a Metropolis algorithm (an approximate Gibbs sampler) for two-species competition dynamics modeled by Lotka-Volterra differential equations, approximated by the classical fourth-order Runge-Kutta method. The Metropolis algorithm (as opposed to a Gibbs sampler) is necessary due to nonlinearity and truncated normal data and process models. Proposals are generated from an approximate Gibbs sampler without appropriate normalizing constants.

```
 LVrkcomp <- function(y,s2epmn,s2epvar,s2etamn,s2etavar,aug,ngibbs,
    upper=10000,,augIN=TRUE){
     ### y is the data array, T x n x 2
     ### s2epmn and s2epvar:  mean and variance of data error
     ### s2etamn and s2etavar:  mean and variance of process error
     ### aug: quantifies the process augmentation; aug=1/dt, reciprocal
                  change in time between process model predictions.
     ### ngibbs: number of MCMC iterations
     ### upper: upper truncation bound

     ###
     ###  Subroutines
     ###

source("invgammastrt.R")     ### finds parameters of Inverse Gamma
source("rkcomp.R")             ### RK4
require(msm)
require(bayesm)

logdIG <- function(s2,q,r){
  -(q+1)*log(s2)-1/(r*s2)-q*log(r)-lgamma(q)
}

dtnorm <- function(x, mean=0,sd=1, lower=-Inf,upper=Inf,log=FALSE){
    ret <- numeric(length(x))
    ret[x < lower | x > upper] <- 0
    ret[is.na(x)] <- NA
    ind <- (x >= lower & x <= upper & !is.na(x))
    if (any(ind)) {
```

```
        denom <- pnorm(upper, mean, sd) - pnorm(lower, mean,
            sd)
        xtmp <- dnorm(x, mean, sd, log)
        if (log)
            xtmp <- xtmp - log(denom)
        else xtmp <- xtmp/denom
        ret[x >= lower & x <= upper & !is.na(x)] <- xtmp[ind]
    }
    ret
}

logdtnorm <- function (x, mean = 0, sd = 1){
    lower=0
    upper=Inf
    ret <- numeric(length(x))
    ret[x < lower | x > upper] <- 0
    ret[is.na(x)] <- NA
    ind <- (x >= lower & x <= upper & !is.na(x))
    if (any(ind)) {
        xtmp <- dnorm(x,mean,sd,log=TRUE)-
                        pnorm(mean/sd,log=TRUE)
        ret[x >= lower & x <= upper & !is.na(x)] <- xtmp[ind]
    }
    ret
}

logdtnormfory <- function(x, mean = 0, sd = 1){
    upper=Inf
    lower=0
    ret <- matrix(NA,length(mean),dim(x)[2])
    ret[x < lower | x > upper] <- 0
    ret[is.na(x)] <- NA
    ind <- (x >= lower & x <= upper & !is.na(x))
    if (any(ind)) {
        xtmp <- dnorm(x,mean,sd,log=TRUE)-
                       matrix(pnorm(mean/sd,
                       log=TRUE),length(mean),dim(x)[2])
        ret[x >= lower & x <= upper & !is.na(x)] <- xtmp[ind]
    }
    ret
}

dtnormfory <- function(x,mean=0,sd=1,
                               lower=-Inf,upper=Inf,log=FALSE){
    ret <- matrix(0,length(mean),dim(x)[2])
    ret[x < lower | x > upper] <- 0
    ret[is.na(x)] <- NA
    ind <- (x >= lower & x <= upper & !is.na(x))
    if (any(ind)) {
        denom <- matrix(pnorm(upper, mean, sd) -
                          pnorm(lower, mean, sd),
```

```
                          length(mean),dim(x)[2])
        xtmp <- dnorm(x, mean, sd, log)
        if (log)
            xtmp <- xtmp - log(denom)
        else xtmp <- xtmp/denom
        ret[x >= lower & x <= upper & !is.na(x)] <- xtmp[ind]
    }
    ret
}

getgvec <- function(f,sig,n){
     ### Approximates bias correction for truncated normal
     ### See Appendix A
  l=-100
  N=length(f)
  gtmp=seq(l,max(f),,n)
  G=gtmp+sig*(exp(dnorm(gtmp/sig,log=TRUE)-
                          pnorm(gtmp/sig,log=TRUE)))
  while(min(G)>=min(f)){
    l=l-100
    gtmp=c(1,gtmp)
    n=n+1
    G=gtmp+sig*(exp(dnorm(gtmp/sig,log=TRUE)-
                          pnorm(gtmp/sig,log=TRUE)))
  }
  diffs=outer(as.vector(G),as.vector(f),FUN="-")
  diffs.pos=diffs
  diffs.pos[diffs < 0]=NA
  diffs.neg=diffs
  diffs.neg[diffs >= 0]=NA
  idxu.mat=(diffs.pos==
              t(apply(diffs.pos,2,min,na.rm=TRUE)%x%matrix(1,1,n)))
  idxl.mat=(diffs.neg==
              t(apply(diffs.neg,2,max,na.rm=TRUE)%x%matrix(1,1,n)))
  idxu.mat[is.na(idxu.mat)]=FALSE
  idxl.mat[is.na(idxl.mat)]=FALSE
  G1=(G%x%matrix(1,1,N))[idxl.mat]
  G2=(G%x%matrix(1,1,N))[idxu.mat]
  g1=(gtmp%x%matrix(1,1,N))[idxl.mat]
  g2=(gtmp%x%matrix(1,1,N))[idxu.mat]
  beta1=(G2-G1)/(g2-g1)
  g=(f-G1)/beta1+g1
  g
}

rtn <- function(mu,sig){
  mu + sig*qnorm(log(runif(length(mu)))+
  pnorm(mu/sig,log=TRUE),lower.tail=FALSE,log.p=TRUE)
}

dataaug <- function(y,aug){
```

```
  T=dim(y)[1]
  T.aug=(T-1)*aug+1
  y.aug=array(NA,c(T.aug,dim(y)[2],dim(y)[3]))
  for(t in 1:T){
    y.aug[(t-1)*aug+1,,]=y[t,,]
  }
  y.aug
}

   ###
   ###  Augment Data
   ###

y=dataaug(y,aug)

   ###
   ###  Setup Variables
   ###

T=dim(y)[1]
n=dim(y)[2]
nt=apply(!is.na(y),1,sum)
T2=T+2
dt=1/aug
nprec=100

dataTF=matrix(TRUE,T,2)
if(augIN==FALSE){
  dataTF=!apply(is.na(y),c(1,3),all)
}

xsave=array(0,c(T2,ngibbs,2))
thetasave=matrix(0,6,ngibbs)
s2epsave=matrix(0,1,ngibbs)
s2etasave=matrix(0,2,ngibbs)

   ###
   ###  Hyperpriors and Constants
   ###

rep=invgammastrt(s2epmn,s2epvar)$r
qep=invgammastrt(s2epmn,s2epvar)$q
reta=invgammastrt(s2etamn,s2etavar)$r
qeta=invgammastrt(s2etamn,s2etavar)$q

mu0=matrix(c(10,10),2,1)
s20=100
mu0=getgvec(mu0,sqrt(s20),nprec)
mua=.5
s2a=1
mub=40
```

```
s2b=1000

mutheta=matrix(c(.6762,564.4,1,1.068,202.2,1),6,1)
    ### prior means for process model parameters
s2theta=matrix(c(.02836,708.5,10,.2610,583.2,10),6,1)
    ### prior variances for process model parameters

   ###
   ###  Initial Values
   ###

s2ep=1500
s2eta1=600
s2eta2=200
s2eta=c(s2eta1,s2eta2)
x0=y[1,1,]-.01
x00=x0-.01

a1=.2
b1=mean(y[T,,1],na.rm=TRUE)
a2=.2
b2=mean(y[T,,2],na.rm=TRUE)
theta=mutheta
thetalow=matrix(c(0,0,0,0,0,0),6,1)
thetahigh=matrix(c(100000,100000,100000,100000,
                                    100000,100000),6,1)
mhep=1
mheta=1
mhb=1
mha=1
mhtheta=1
mhx=matrix(0,T+2,2)
btune=5
atune=.5
thetatune=matrix(c(.2,10,.2,.2,10,.2),6,1)
eptune=500
etatune=200
x0tune=2
#xttune=2
xttune=20

x=rbind(x00,x0,matrix(0,T,2))
for(t in 1:T){
  x[2+t,]=rkcomp(theta,dt,x[2+t-1,],1)$x[,2]
}
x=x+1

   ###
   ### Initialize timing variables
   ###
```

```
time1=proc.time()
time2=time1
timeidx=0

   ###
   ###  Begin Metropolis Loop
   ###

for(k in 2:ngibbs){
  cat(k," ")

   ###
   ### Timing Calculations
   ###

  if(k==12){
    tentime=(proc.time()-time1)[3]
    cat("\n",tentime*(ngibbs/600),"expected minutes","\n")
  }
  if(k%%100==0){
    timeidx=timeidx+1
    elapsetime=(proc.time()-time2)[3]
    cat("\n",elapsetime/60,"elapsed minutes"," ")
    leftidx=ngibbs/100 - timeidx
    cat(" ",(elapsetime/timeidx)*leftidx/60,"remaining minutes","\n")
  }

   ###
   ###  Sample x_0
   ###

  cat("x0"," ")

  g0kminus1=getgvec(matrix(x[2+0,],ncol=1),x0tune,nprec)
  x0star=rtn(g0kminus1,x0tune)
  g0star=getgvec(x0star,x0tune,nprec)

  f1star=matrix(rkcomp(theta,dt,x0star,1)$x[,2],2,1)
  g1star1=getgvec(f1star[1,],sqrt(s2eta[1]),nprec)
  g1star2=getgvec(f1star[2,],sqrt(s2eta[2]),nprec)
  g1star=rbind(g1star1,g1star2)

  f1=matrix(rkcomp(theta,dt,x[2+0,],1)$x[,2],2,1)
  g11=getgvec(f1[1,],sqrt(s2eta[1]),nprec)
  g12=getgvec(f1[2,],sqrt(s2eta[2]),nprec)
  g1=rbind(g11,g12)


  mhratio1=sum(logdtnorm(x[2+1,],g1star,sqrt(s2eta)))+
                      logdtnorm(x0star,mu0,sqrt(s20))+
                      logdtnorm(x[2+0,],g0star,x0tune)
```

```
mhratio2=sum(logdtnorm(x[2+1,],g1,sqrt(s2eta)))+
                    logdtnorm(x[2+0,],mu0,sqrt(s20))+
                    logdtnorm(x0star,g0kminus1,x0tune)

mhratio=exp(mhratio1-mhratio2)
updateTF=(mhratio > runif(2))
x[2+0,updateTF]=x0star[updateTF]
mhx[2+0,updateTF]=mhx[2+0,updateTF]+1

 ###
 ###  Sample x_t
 ###

cat("xt"," ")

   ###
   ###  getting proposals for x_t
   ###

gtkminus1.mat=matrix(getgvec(matrix(x[-(1:2),],ncol=1),
                             xttune,nprec),ncol=2)
xtstar.mat=matrix(rtn(as.vector(gtkminus1.mat),xttune),
                              ncol=2)
gtstar.mat=matrix(getgvec(matrix(xtstar.mat,ncol=1),
                               xttune,nprec),ncol=2)
  ###
  ###  evaluating proposals for x_t via Metropolis-Hastings
  ###

for(t in 1:(T-1)){

   gtkminus1=gtkminus1.mat[t,]
   xtstar=xtstar.mat[t,]
   gtstar=gtstar.mat[t,]

  gtystar=getgvec(xtstar,sqrt(s2ep),nprec)
  gtykminus1=getgvec(matrix(x[2+t,],2,1),sqrt(s2ep),nprec)

  ftplus1star=matrix(rkcomp(theta,dt,xtstar,1)$x[,2],2,1)
  if(!any(ftplus1star < 0 || ftplus1star > upper)){

   gtplus1star1=getgvec(ftplus1star[1,],sqrt(s2eta[1]),nprec)
   gtplus1star2=getgvec(ftplus1star[2,],sqrt(s2eta[2]),nprec)
   gtplus1star=rbind(gtplus1star1, gtplus1star2)


    ftplus1kminus1=matrix(rkcomp(theta,dt,x[2+t,],1)$x[,2],2,1)
    gtplus1kminus11=getgvec(ftplus1kminus1[1,],sqrt(s2eta[1]),nprec)
    gtplus1kminus12=getgvec(ftplus1kminus1[2,],sqrt(s2eta[2]),nprec)
    gtplus1kminus1=rbind(gtplus1kminus11,gtplus1kminus12)
```

```
    ftk=matrix(rkcomp(theta,dt,x[2+t-1,],1)$x[,2],2,1)
    gtk1=getgvec(ftk[1,],sqrt(s2eta[1]),nprec)
    gtk2=getgvec(ftk[2,],sqrt(s2eta[2]),nprec)
    gtk=rbind(gtk1,gtk2)

    mhratio1=apply(logdtnormfory(t(y[t,,]),gtystar,sqrt(s2ep)),1,
                  sum,na.rm=TRUE)+logdtnorm(x[2+t+1,],
                  gtplus1star,sqrt(s2eta))+logdtnorm(xtstar,
                  gtk,sqrt(s2eta))+logdtnorm(x[2+t,],gtstar,xttune)
    mhratio2=apply(logdtnormfory(t(y[t,,]),gtykminus1,sqrt(s2ep)),1,
                   sum,na.rm=TRUE)+logdtnorm(x[2+t+1,],
                   gtplus1kminus1,sqrt(s2eta))+logdtnorm(x[2+t,],
                   gtk,sqrt(s2eta))+logdtnorm(xtstar,gtkminus1,xttune)

    mhratio=exp(mhratio1-mhratio2)
    updateTF=(mhratio > runif(2))
    x[2+t,updateTF]=xtstar[updateTF]
    mhx[2+t,updateTF]=mhx[2+t,updateTF]+1

  }
}

 ###
 ###  Sample x_T
 ###

cat("xT"," ")

gTkminus1=getgvec(x[2+T,],xttune,nprec)
xTstar=matrix(rtn(gTkminus1,xttune),2,1)
gTstar=getgvec(xTstar,xttune,nprec)

gTystar=getgvec(xTstar,sqrt(s2ep),nprec)
gTykminus1=getgvec(matrix(x[2+T,],2,1),sqrt(s2ep),nprec)

fTk=matrix(rkcomp(theta,dt,x[2+T-1,],1)$x[,2],2,1)
gTk1=getgvec(fTk[1,],sqrt(s2eta[1]),nprec)
gTk2=getgvec(fTk[2,],sqrt(s2eta[2]),nprec)
gTk=rbind(gTk1,gTk2)

mhratio1=apply(logdtnormfory(t(y[T,,]),gTystar,sqrt(s2ep)),1,
                sum,na.rm=TRUE)+logdtnorm(xTstar,gTk,sqrt(s2eta))+
                logdtnorm(x[2+T,],gTstar,xttune)
mhratio2=apply(logdtnormfory(t(y[T,,]),gTykminus1,sqrt(s2ep)),1,
                sum,na.rm=TRUE)+logdtnorm(x[2+T,],gTk,sqrt(s2eta))+
                logdtnorm(xTstar,gTkminus1,xttune)

mhratio=exp(mhratio1-mhratio2)
updateTF=(mhratio > runif(2))
x[2+T,updateTF]=xTstar[updateTF]
```

```
mhx[2+T,updateTF]=mhx[2+T,updateTF]+1

 ###
 ###  Sample theta
 ###

cat("theta"," ")

thetastar=rtrun(theta,thetatune,thetalow,thetahigh)

ftstarall=matrix(0,T,2)
gtstarall=matrix(0,T,2)
ftkall=matrix(0,T,2)
gtkall=matrix(0,T,2)
for(t in 1:T){
  ftstarall[t,]=rkcomp(thetastar,dt,x[2+t-1,],1)$x[,2]
  ftkall[t,]=rkcomp(theta,dt,x[2+t-1,],1)$x[,2]
  if(!any(ftstarall < 0)){
    gtstarall[t,]=getgvec(matrix(ftstarall[t,],2,1),sqrt(s2eta),nprec)
  }
  gtkall[t,]=getgvec(matrix(ftkall[t,],2,1),sqrt(s2eta),nprec)
}

mhratio1=sum(logdtnorm(x[3:(T+2),1][dataTF[,1]],(gtstarall)[,1]
               [dataTF[,1]],sqrt(s2eta[1])),na.rm=TRUE)+
               sum(logdtnorm(x[3:(T+2),2][dataTF[,2]],(gtstarall)[,2]
               [dataTF[,2]],sqrt(s2eta[2])),na.rm=TRUE)+
               sum(dtnorm(thetastar,mutheta,sqrt(s2theta),thetalow,
               thetahigh,log=TRUE))+sum(dtnorm(theta,thetastar,
               thetatune,thetalow,thetahigh,log=TRUE))
mhratio2=sum(logdtnorm(x[3:(T+2),1][dataTF[,1]],(gtkall)[,1]
               [dataTF[,1]],sqrt(s2eta[1])),na.rm=TRUE)+
               sum(logdtnorm(x[3:(T+2),2][dataTF[,2]],(gtkall)[,2]
               [dataTF[,2]],sqrt(s2eta[2])),na.rm=TRUE)+
               sum(dtnorm(theta,mutheta,sqrt(s2theta),thetalow,
               thetahigh,log=TRUE))+sum(dtnorm(thetastar,theta,
               thetatune,thetalow,thetahigh,log=TRUE))
mhratio=exp(mhratio1-mhratio2)

if(mhratio > runif(1)){
  theta=thetastar
  mhtheta=mhtheta+1
}

 ###
 ###  Sample s2ep
 ###

cat("s2ep"," ")

s2epstar=rtnorm(1,s2ep,eptune,0)
```

```
gtystarall=matrix(getgvec(matrix(x[-(1:2),],ncol=1),
               sqrt(s2epstar),nprec),T,2)
gtykminus1all=matrix(getgvec(matrix(x[-(1:2),],ncol=1),
               sqrt(s2ep),nprec),T,2)

mhratio1=sum(logdtnorm(as.vector(y[1:T,,1]),rep(gtystarall[,1],n),
               sqrt(s2epstar)),na.rm=TRUE)+
               sum(logdtnorm(as.vector(y[1:T,,2]),rep(gtystarall[,2],n),
               sqrt(s2epstar)),na.rm=TRUE)+logdIG(s2epstar,qep,rep)+
               dtnorm(s2ep,s2epstar,eptune,0,log=TRUE)
mhratio2=sum(logdtnorm(as.vector(y[1:T,,1]),rep(gtykminus1all[,1],n),
               sqrt(s2ep)),na.rm=TRUE)+
               sum(logdtnorm(as.vector(y[1:T,,2]),rep(gtykminus1all[,2],n),
               sqrt(s2ep)),na.rm=TRUE)+logdIG(s2ep,qep,rep)+
               dtnorm(s2epstar,s2ep,eptune,0,log=TRUE)
mhratio=exp(mhratio1-mhratio2)

if(mhratio > runif(1)){
  s2ep=s2epstar
  mhep=mhep+1
}

 ###
 ###  Sample s2eta
 ###

cat("s2eta"," ")

s2etastar=rtnorm(2,s2eta,etatune,0)

gtstarall1=getgvec(ftkall[,1],sqrt(s2etastar[1]),nprec)
gtstarall2=getgvec(ftkall[,2],sqrt(s2etastar[2]),nprec)
gtstarall=cbind(gtstarall1,gtstarall2)

gtkminus1all1=getgvec(ftkall[,1],sqrt(s2eta[1]),nprec)
gtkminus1all2=getgvec(ftkall[,2],sqrt(s2eta[2]),nprec)
gtkminus1all=cbind(gtkminus1all1,gtkminus1all2)

mhratio1=sum(logdtnorm(x[3:(T+2),1][dataTF[,1]],(gtstarall)[,1]
              [dataTF[,1]],sqrt(s2etastar[1])),na.rm=TRUE)+
              sum(logdtnorm(x[3:(T+2),2][dataTF[,2]],(gtstarall)[,2]
              [dataTF[,2]],sqrt(s2etastar[2])),na.rm=TRUE)+
              sum(logdIG(s2etastar,qeta,reta))+
              sum(dtnorm(s2eta,s2etastar,etatune,0,log=TRUE))
mhratio2=sum(logdtnorm(x[3:(T+2),1][dataTF[,1]],(gtkminus1all)[,1]
              [dataTF[,1]],sqrt(s2eta[1])),na.rm=TRUE)+
              sum(logdtnorm(x[3:(T+2),2][dataTF[,2]],(gtkminus1all)[,2]
              [dataTF[,2]],sqrt(s2eta[2])),na.rm=TRUE)+
              sum(logdIG(s2eta,qeta,reta))+
              sum(dtnorm(s2etastar,s2eta,etatune,0,log=TRUE))
```

```
  mhratio=exp(mhratio1-mhratio2)

  if(mhratio > runif(1)){
    s2eta=s2etastar
    mheta=mheta+1
  }

  ###
  ###  Save Samples
  ###

  xsave[,k,]=x
  thetasave[,k]=theta
  s2epsave[,k]=s2ep
  s2etasave[,k]=s2eta

cat("\n")
}  #### END MH LOOP ####

###
###  Write output
###

list(xsave=xsave,thetasave=thetasave,s2epsave=s2epsave,
s2etasave=s2etasave,ngibbs=ngibbs,mhtheta=mhtheta,
mhx=mhx,mhep=mhep,mheta=mheta,y=y,n=n,augIN=augIN)
}
```